Efficient Two-Phase Approaches for Branch-and-Bound Style Resource Constrained Scheduling

Mingsong Chen*, Fan Gu, Lei Zhou, Geguang Pu and Xiao Liu

Shanghai Key Lab of Trustworthy Computing

East China Normal University, China

*** Presenter**

January 7, 2014

Outline

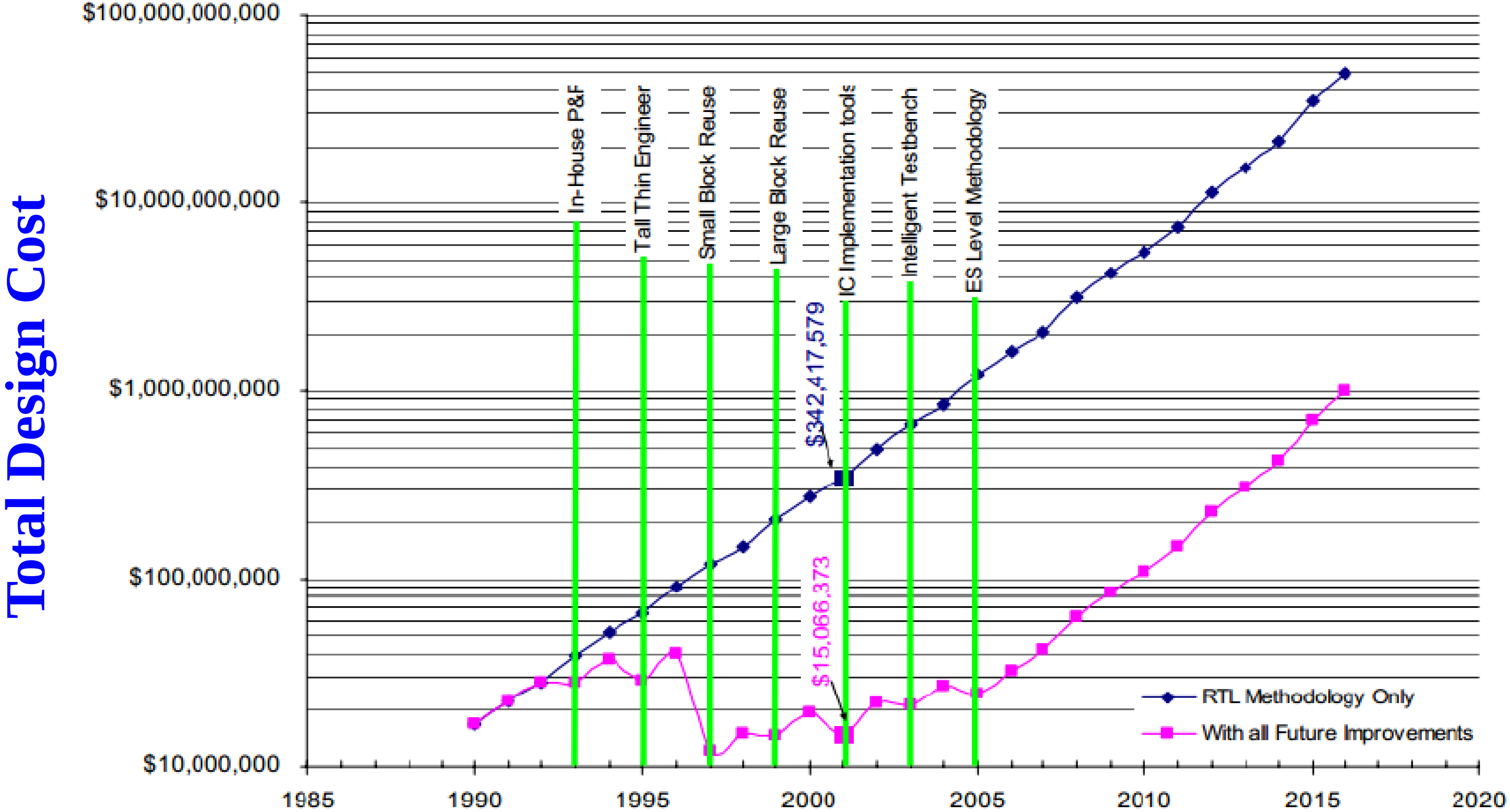
- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Two-Phase Approaches
  - Bounded-Operation Approach
  - Non-Chronological Backtrack
  - Search Space Speculation
- Experiments
- Conclusion

Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Two-Phase Approaches
  - Bounded-Operation Approach
  - Non-Chronological Backtrack
  - Search Space Speculation
- Experiments
- Conclusion

SoC Design Cost Model

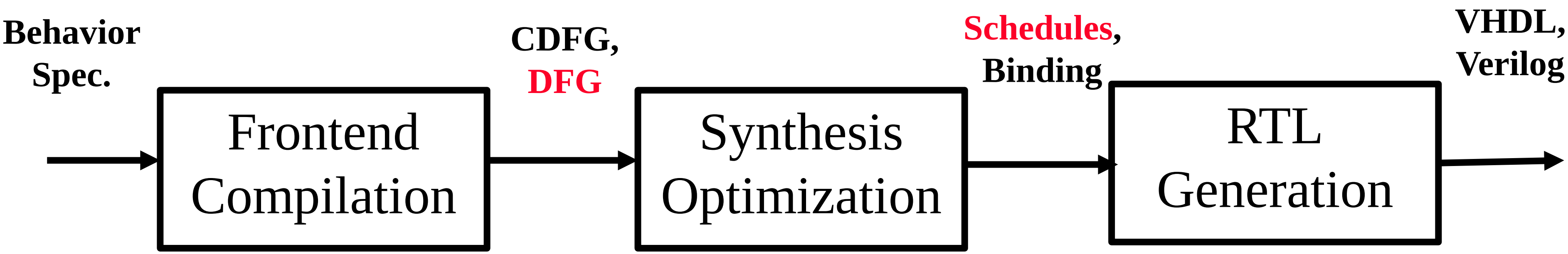
Big Savings by using ESL Methodology



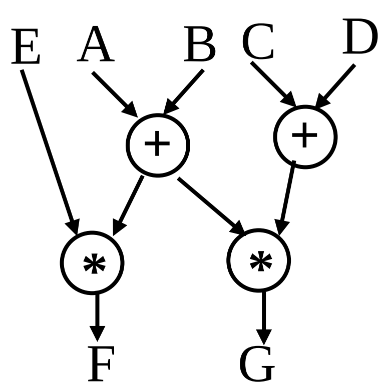
Rising cost of IC design and effect of CAD tools
(Courtesy: Andrew Kahng, UCSD and SRC)

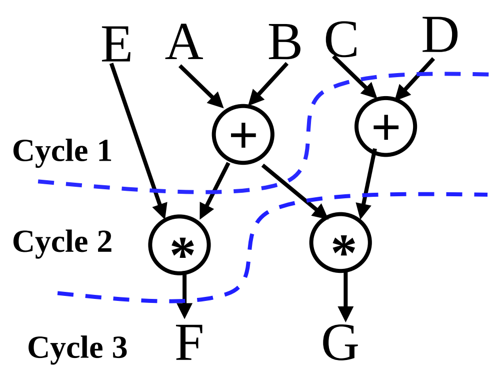
High Level Synthesis

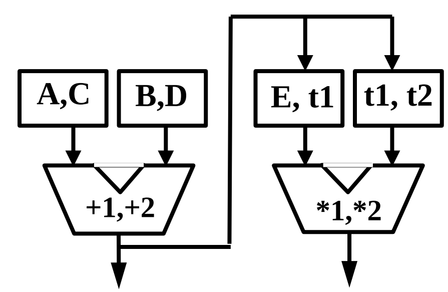
- **Convert ESL specification to RTL implementation, and satisfy the design constraints.**
  - **Input:** Behavior specifications (C, SystemC, etc.), and design constraints (delay, power, area, etc.)
  - **Output:** RTL implementations (datapath, controller)




```
int Sample(){
var  A,B,C,D,E,F,G : int;
Read(A, B, C, D, E);
F = E * (A + B);
G = (A + B) * (C + D);
……
}
```

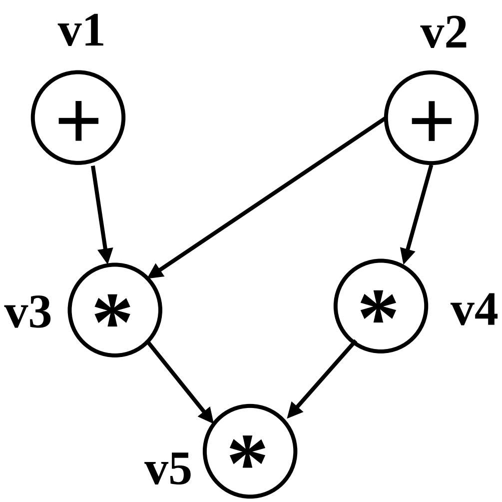




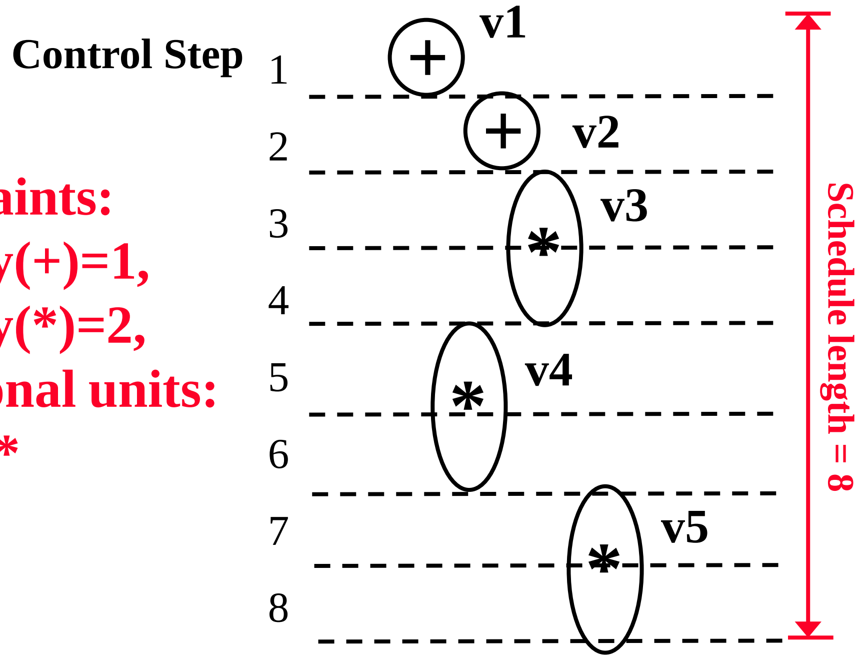


Resource Constrained Scheduling

- **Scheduling is a mapping of operations to control steps**
  - **Given a DFG and a set of resource constraints, RCS tries to find a (optimal) schedule with minimum overall control steps.**
- **Various resource constraints (e.g., functional units, power, …).**



Constraints:
Delay(+)=1,
Delay(*)=2,
Functional units:
1+, 1*



RCS is NP-Complete. RCS should take care of
1) Operation precedence. 2) Resource sharing constraints

Basic Solutions

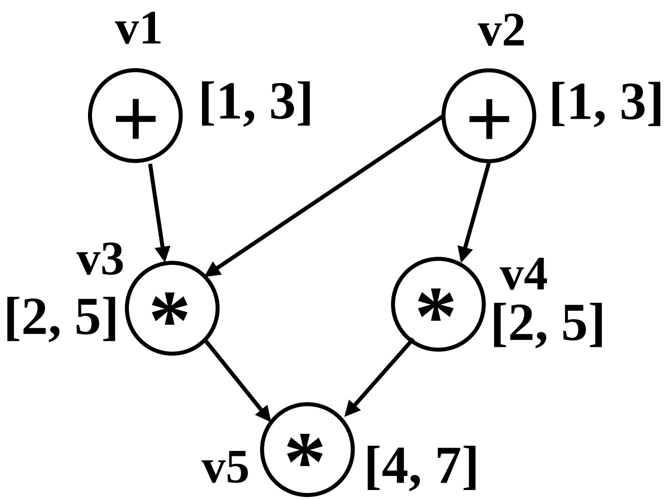
- Non-optimal heuristics
  - Force Directed Scheduling
  - List scheduling
    - ✓ Pros: Fast to get near-optimal results
    - ✓ Cons: schedules may not be tight
- Optimal sequential approaches
  - Integer linear programming (ILP)
    - ✓ Pros: easy modeling
    - ✓ Cons: scalability, cannot handle non-integer time
  - Branch-and-bound
    - ✓ Pros: can prune the fruitless search space efficiently
    - ✓ Cons: hard to achieve a tight initial upper-bound

Outline

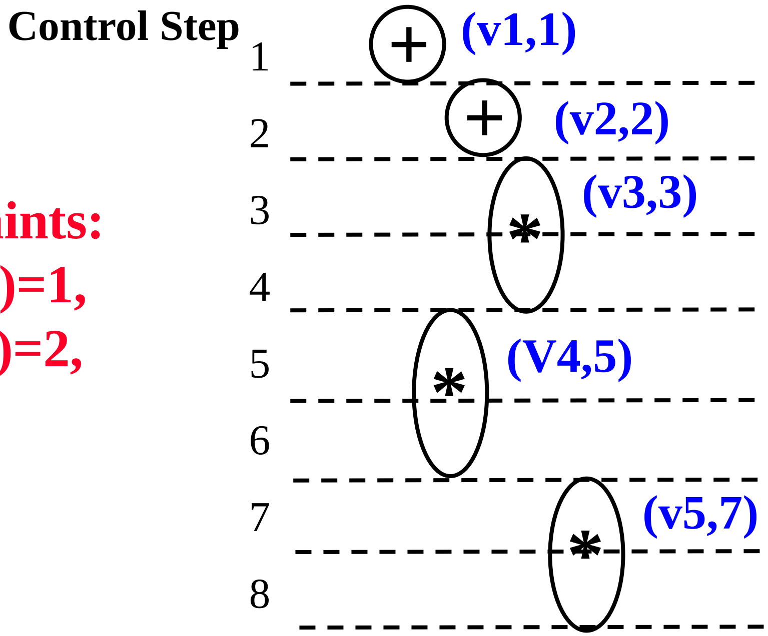
- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Two-Phase Approaches
  - Bounded-Operation Approach
  - Non-Chronological Backtrack
  - Search Space Speculation
- Experiments
- Conclusion

Scheduling Using [ASAP, ALAP]

- **Based on [ASAP, ALAP], naively enumerating all the possibilities can be extremely time consuming**
  - **The operations are enumerated in a specific order**
  - **Each operation is enumerated from ASAP to ALAP**



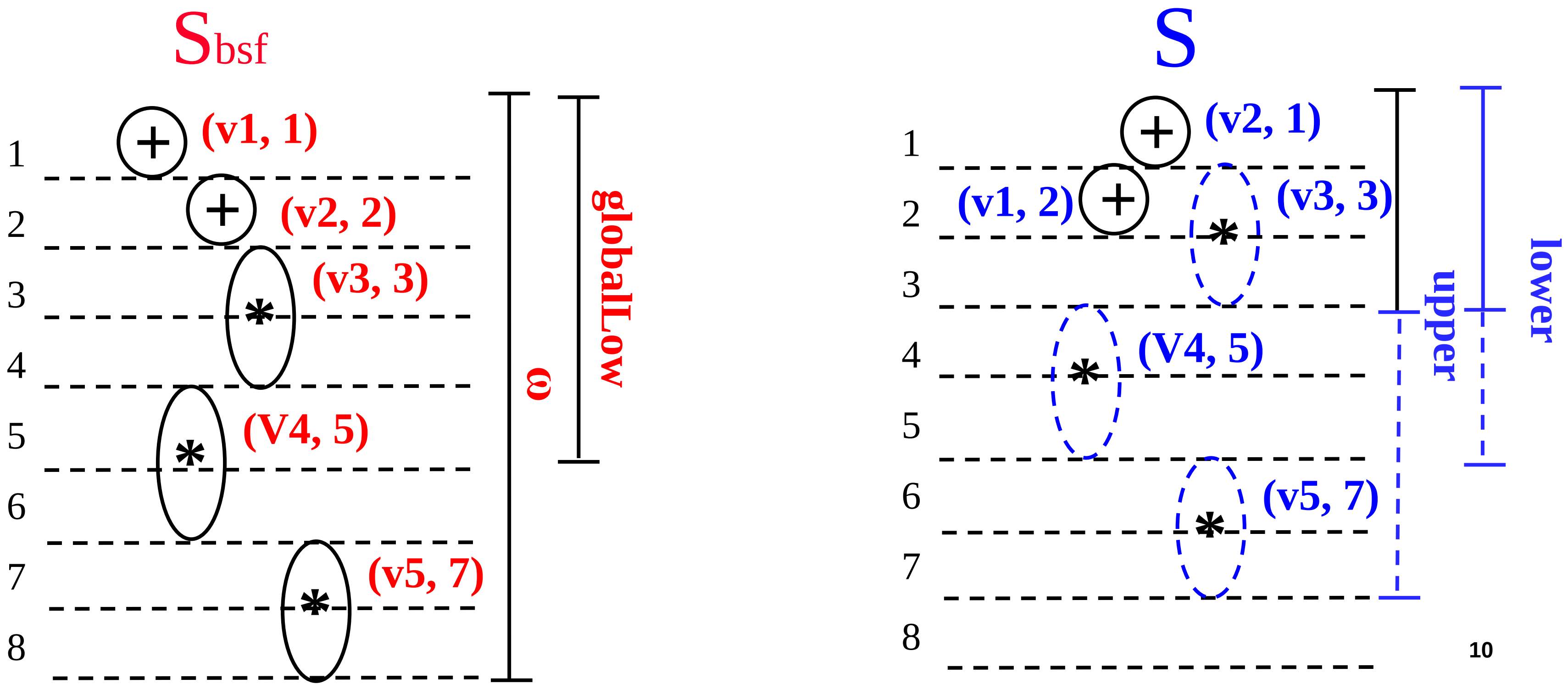
Constraints:
Delay(+)=1,
Delay(*)=2,
1+, 1*



- **A schedule is a binary relation of operations and corresponding dispatching control step**
  - **E.g., {(v1, 1), (v2, 2), (v3, 3), (v4, 5), (v5, 7)}**

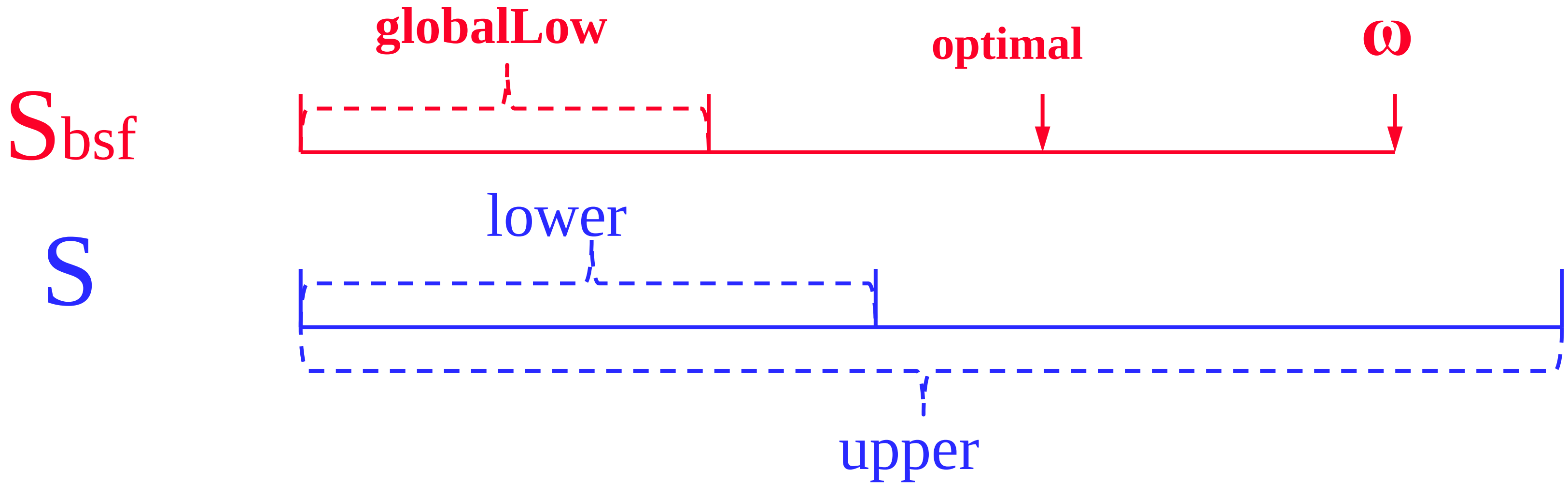
Branch and Bound Style RCS (BULB)

- BULB tries to prune fruitless enumerations.
- B&B approach keeps two data structure regarding bound information.
  - S_{bsf} , best complete schedule searched so far
  - S, current incomplete schedule



Pruning in BULB

- **Pruning [lower > ω] → Backtrack to last operation**
- **Termination [globalLow == ω or fully explored]**
- **Substitution [if(upper < ω) ω = upper]**



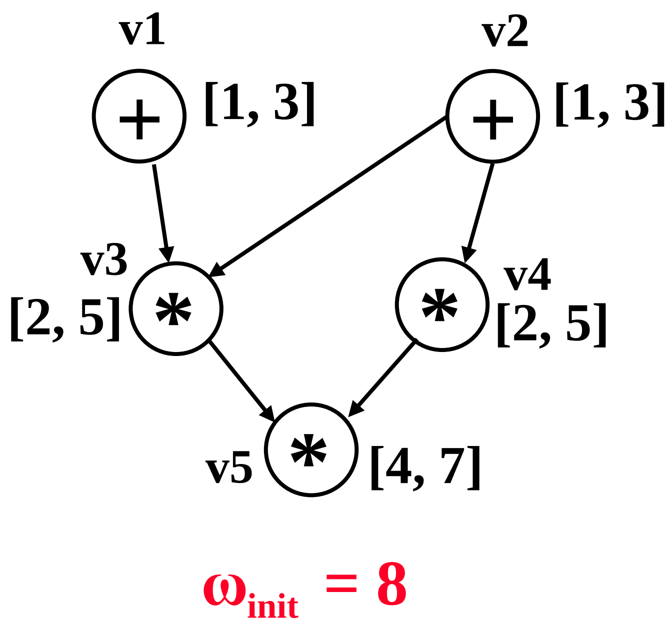
ω plays an important role in B&B approaches. A smaller ω can
- tighten the [ASAP, ALAP] intervals, i.e., search space;
- enable the fast pruning of inferior schedules during RCS.

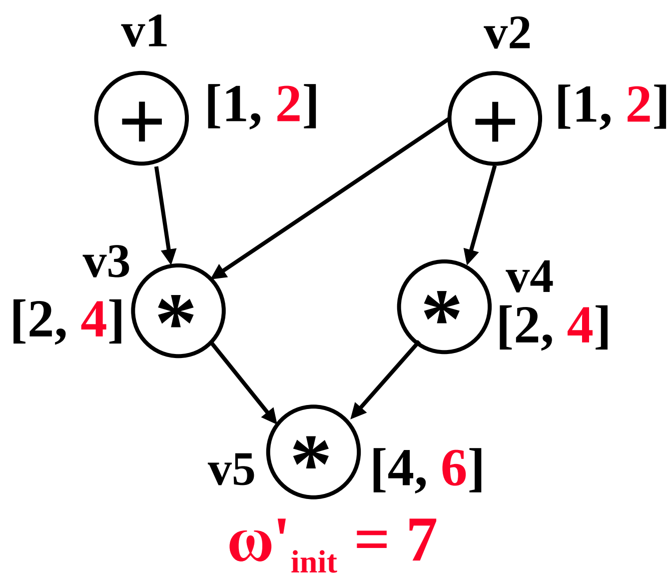
Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Two-Phase Approaches
  - Bounded-Operation Approach
  - Non-Chronological Backtrack
  - Search Space Speculation
- Experiments
- Conclusion

Importance of Initial Feasible Schedule

- ALAP(OPi, Sinit) = ω_{init} – CPw(G(OP_i)), where Sinit is an initial feasible schedule, and ω_{init} = length(S_{init}) .

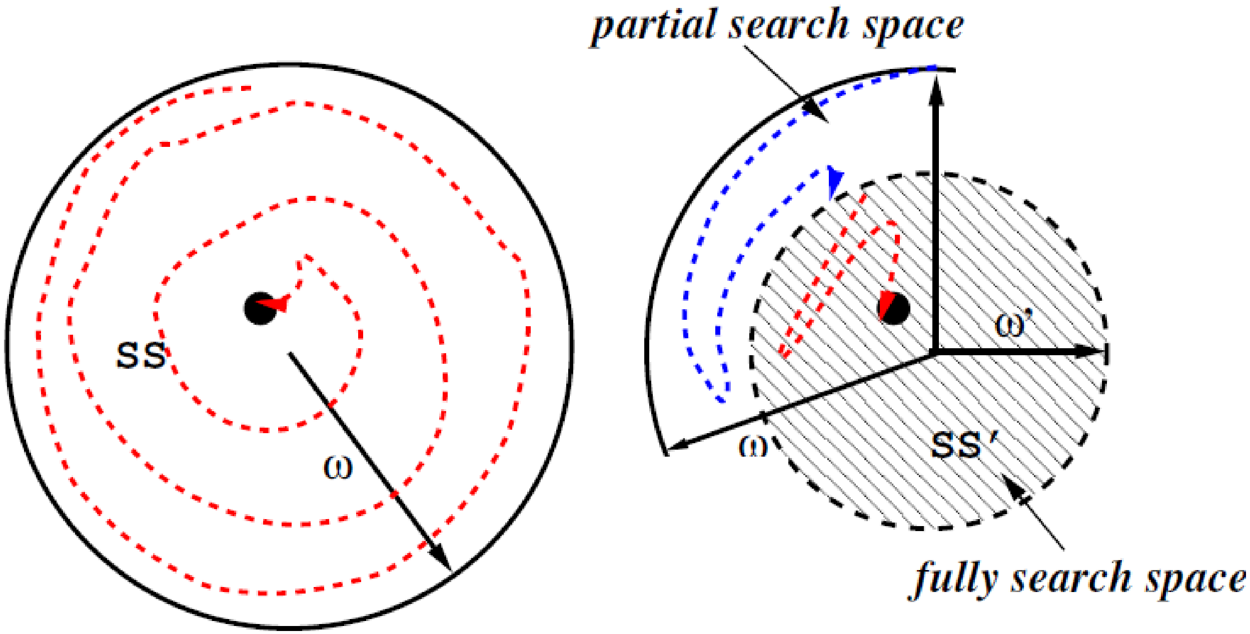




- List scheduling cannot always guarantee a small ω_{init}, since it only considers only one possible schedule combination of unscheduled operations.
- How to quickly find a small ω_{init} is a key issue in RCS.

Basic Idea of Our 2P Approach

- **Two-phase approach has two steps**
  - **Step 1 does partial search on the search space coarsely to achieve a tight schedule.**
  - **Step 2 fully scans the search space in the same way as BULB approach, but with a tight ω_{init} achieved from step 1.**

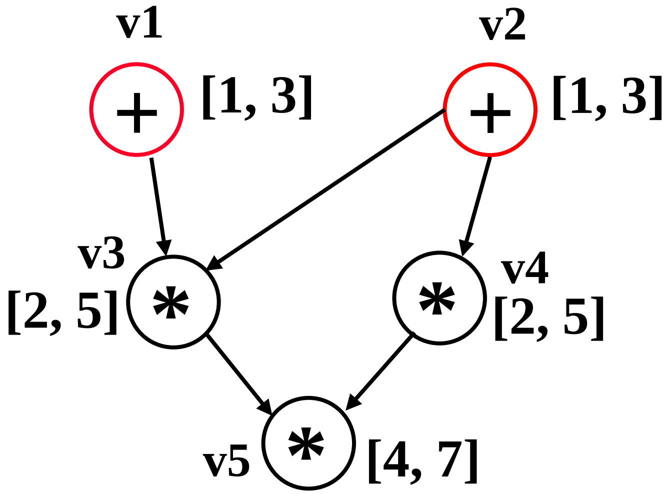


a) BULB Searching b) Two−phase Searching

- **Partial Search should achieve a small ω_{init} with small overhead.**

Bounded Operation Approach

- **Basic idea: Less operations involved in partial search.**
- **Bounded operation approach only considers the input nodes. The remaining nodes are estimated using list scheduling approach.**
- **Example:**



$S_1 = \{(OP_1, 1)\}$

$S_2 = \{(OP_1, 1), \{(OP2, 2)\}$

ListScheduling(S_1) = ListScheduling(S_2) = 8

$S_3 = \{(OP_1, 1), \{(OP_2, 3)\}$

ListScheduling(S_3) = 9

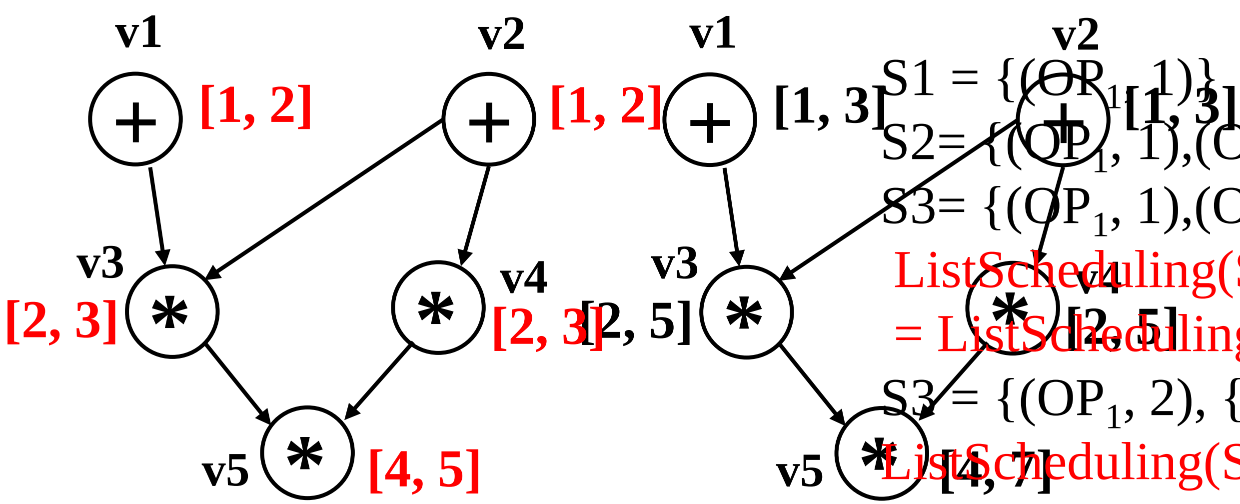
$S_4 = \{(OP_1, 2), \{(OP_2, 1)\}$

ListScheduling(S_4) = 7 = Length(S_{opt})

Only 4 tries can achieve the tightest initial schedule.
Bounded operation method can efficiently avoid trap in the deep search.

Search Space Speculation

- **Basic idea: Smaller search range of each operation.**
- **By adopting a greedy strategy, our speculation approach assumes that the global optimal result will be always located in the first half of orginal range.**
- **Example:**



S1 = {(OP_1, 1)}

S2= {(OP_1, 1),(OP_2, 2)}

S3= {(OP_1, 1),(OP_2, 2),(OP_3, 3)}

ListScheduling(S_1) = ListScheduling(S_2) = ListScheduling(S_3) =8

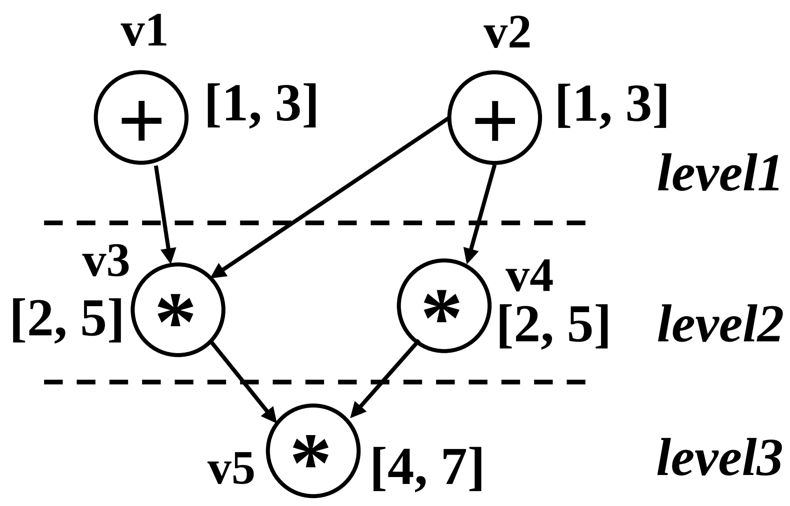
S3 = {(OP_1, 2), {(OP_2, 1)}

ListScheduling(S_3) = 7 = Length(S_{opt})

Only 4 tries can achieve the tightest initial schedule.
Search space speculation can efficiently avoid trap in the deep search.

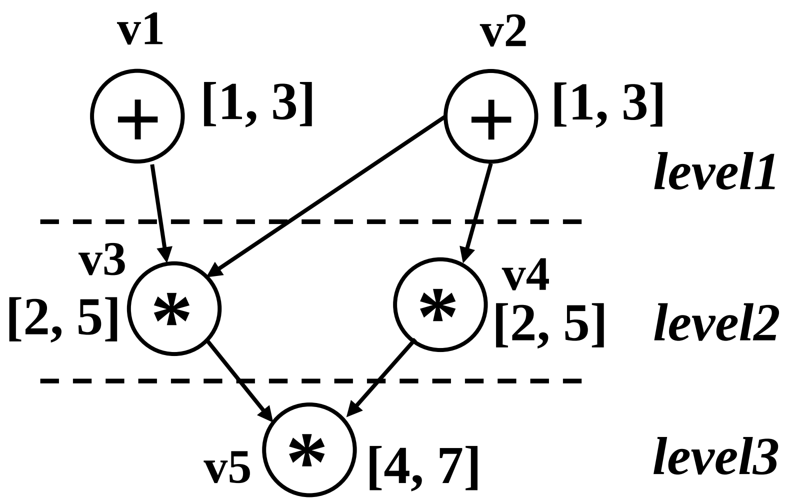
Non-Chronological Backtrack

- **Basic idea: A large backtrack jump to escape the local deep search.**
- **Our non-chronological partial search is based on the DFG level structure.**
  - **Level indicates the precedence between operations.**
  - **Level check condition: All the operations in the i_{th} level are scheduled, and for each operation $op_{i,j}$ in the i_{th} level, $S_{bsf}(op_{i,j}) <= S(op_{i,j})$**



Non-Chronological Backtrack

- **When level check condition holds in the i_{th} level, the scheduling will backtrack to the first dispatched operation of i_{th} level.**
- **Example**



S_{bsf} = { (OP_1, 1), (OP_2, 2), (OP_3, 3), (OP_4, 5), (OP_5, 7)}

S_1 = {(OP_1, 1)}

ListScheduling(S_1) = 8

S_2 = {(OP_1, 1), {(OP_2, 2)}will backtrack due to the level check condition

S_2' = { (OP_1, 2), (OP_2,1)}

ListScheduling(S_2') = 7 = Length(S_{opt})

Only 2 tries can achieve the tightest initial schedule.
Non-chronological backtrack can efficiently escape the deep search.

Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Two-Phase Approaches
  - Bounded-Operation Approach
  - Non-Chronological Backtrack
  - Search Space Speculation
- Experiments
- Conclusion

Benchmarks & Settings

- Using benchmarks from *MediaBench*.
- Implementing BULB & our approach using C++.
- All experiments were conducted on a Linux machine with Intel Xeon 3.3GHz Processor and 8G RAM.
- Setting of functional units:

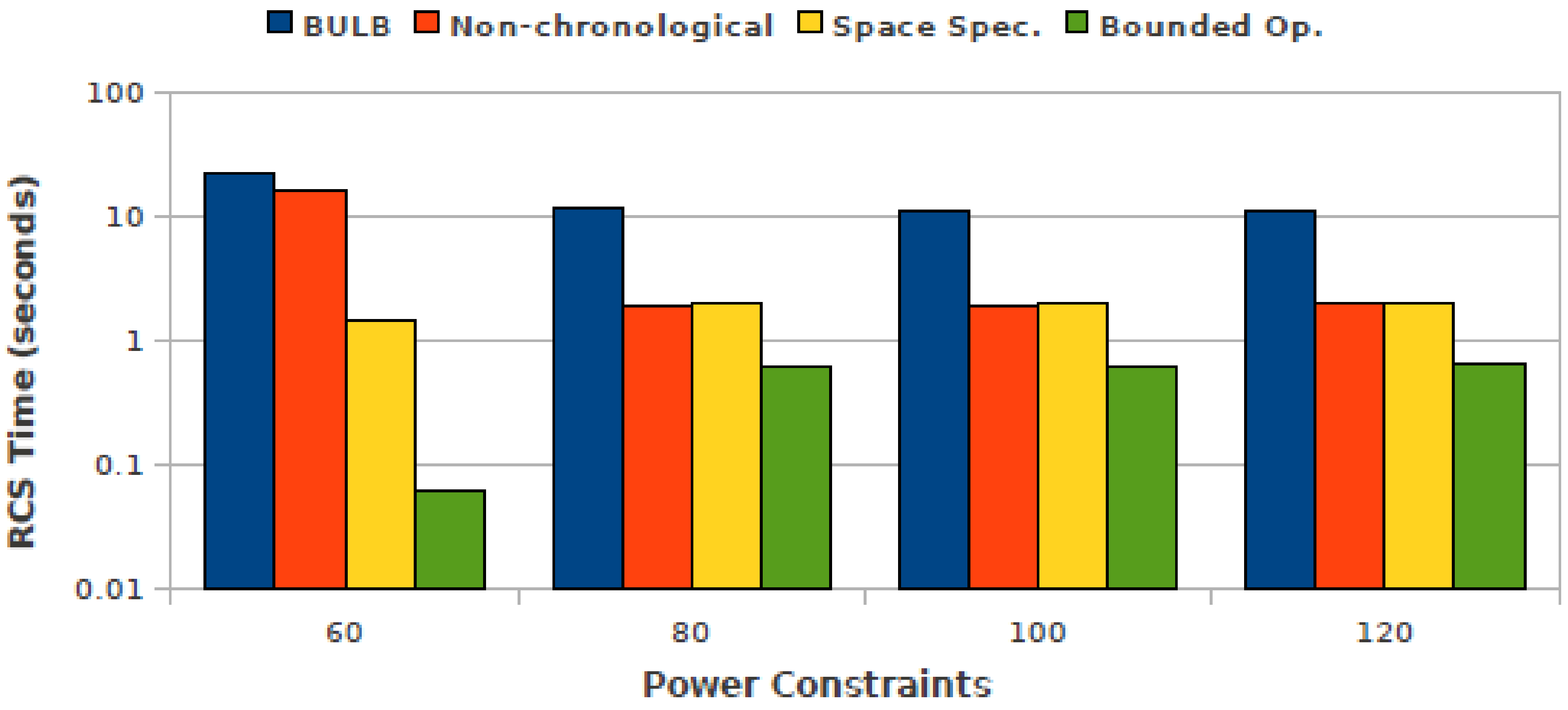
Functional Unit	Operation class	Delay (unit)	Power (unit)	Energy (unit)	Area (unit)
ADD/SUB	+/-	1	**10**	10	**10**
MUL/DIV	*/	2	**20**	40	**40**
MEM	LD/STR	1	**15**	15	**20**
Shift	<</>>	1	**10**	10	**5**
Others	…	1	**10**	10	**10**

Results under Functional Constraints

Design Name	CP	BULB	Bounded Operation		Non-Chronological		Space Speculation		Max Impr.
			T_1	T_{total}	T_1	T_{total}	T_1	T_{total}	
ARFilter	TO	0.16	0.08	0.22	<0.01	0.16	0.10	0.25	1.00
	TO	0.40	0.13	0.49	<0.01	0.40	0.27	0.62	1.00
	TO	0.38	0.14	0.49	<0.01	0.38	0.26	0.62	1.00
	1.40	0.01	<0.01	0.02	<0.01	0.01	<0.01	0.01	1.00
Collapse	TO	TO	162.20	162.20	TO	TO	0.18	0.18	>2.00e4
	TO	TO	TO	TO	TO	TO	TO	TO	NA
Cosine1	TO	63.51	<0.01	<0.01	0.06	0.06	20.94	20.94	1.59e4
	TO	377.58	15.66	15.66	0.03	330.23	21.70	21.70	24.11
	TO	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	1.00
FDCT	TO	20.30	<0.01	<0.01	18.56	18.56	0.26	0.26	5.08e3
	TO	113.92	0.07	0.07	19.87	19.87	0.65	0.65	1.63e3
	TO	11.17	0.63	0.63	1.94	1.94	2.03	2.03	17.73
	TO	2.49	0.03	0.03	0.23	0.23	3.76	3.76	83.00
	TO	0.48	<0.01	<0.01	0.12	0.12	0.16	0.16	120.00
	TO	0.34	0.21	0.21	0.03	0.03	0.14	0.14	11.33
	TO	0.07	0.01	0.01	0.06	0.06	0.02	0.02	7.00
Feedback	TO	85.74	77.63	77.63	77.84	77.84	18.38	18.38	4.66
	TO	TO	TO	TO	265.30	265.30	TO	TO	>13.60
	TO	2.72	2.48	2.48	2.45	2.45	0.58	0.58	4.70

RCS efforts are significantly improved:
- Our 2P approaches outperform both ILP and BULB approaches
- Parallel execution of 2P methods may achieve the best overall performance

Scheduling Using Area of 140 Units



The two-phases approaches (e.g., bounded operation) can achieve a speedup of several orders of magnitude.

Conclusions

- **RCS is a major bottleneck in HLS**
  - Search Branch-and-bound approaches are promising for optimal resource-constrained scheduling
- **Proposed an efficient two-phase B&B approach**
  - Two-phase search space reduction
  - Partial-search heuristics
    - Bounded operation approach, non-chronological backtrack and search space speculation
- **Successfully applied on various benchmarks with different resource constraints**
  - Significant reduction in overall RCS efforts

Thank you !